Representing Many-Body Quantum States using Probabilistic Bits

Daria Mihaila, Leah Heil, Denise Monkau & Jasper Pieterse

January 26, 2024

Abstract

This report introduces probabilistic bits (p-bits), a novel and neuromorphic method, as a tool for representing quantum many-body states. We explore two different applications of p-bits in this context. Firstly, we use p-bits with probabilistic spin logic, as initially proposed by Camsari et al. (2017), to effectively represent smaller quantum systems using truth table representations. Secondly, we examine the role of p-bits in enhancing Gibbs sampling. This enhancement is particularly useful for representing larger quantum many-body systems using neural network quantum states. Our results demonstrate the potential of p-bits in creating truth table representations, giving encouraging results. However, our attempt to use p-bits for speeding up neural network sampling was not fully realised in this project. In summary, while p-bits show promise in representing complex systems, further experiments with actual hardware are necessary to fully understand their effectiveness in sampling processes.

Contents

1 Introduction

The brain performs extraordinarily efficient computation, using at least 4 orders of magnitude less power than the current best neuromorphic silicon circuits [1]. This energy efficiency is a consequence of the severe evolutionary pressure to optimise for power efficiency in the brain [2]. This efficiency manifests in several key ways that differentiate the brain from conventional computers, including the co-location of memory and computation and the selective, sparse information transfer among neurons [3]. Surprisingly, the signals used by our brain seem far from efficient, characterised by high variability and noise [4]. It is hypothesised that this stochasticity might be a strategic energy conservation method, making it a beneficial feature rather than a flaw[5], [6]. This aspect of the brain has inspired the development of probabilistic bits (p-bits).

P-bits are binary units, similar to digital bits, but they randomly fluctuate between 0 and 1. These bits' inherent randomness is useful in stochastic neural networks, such as Boltzmann machines, where p-bits can emulate stochasticity more naturally than traditional methods. Through their neuromorphic design, probabilistic bits offer the prospect of significantly more energy-efficient algorithms [7]. The energy efficiency of p-bits shows promising potential for vast applications through the use of machine learning (ML) in the physical sciences. In recent years, this field has seen remarkable growth, driven by the ability of ML to handle large data sets and complex problems [8]. However, high energy usage of ML algorithms raises concerns about sustainability and forms a bottleneck for further advancements. P-bit hardware architectures promise to alleviate these issues.

This essay explores how p-bits could improve performance specifically within the quantum many-body problem in condensed matter physics. The main objective of this field is to understand the properties of exotic materials by approximating their quantum many-body wavefunctions [9]. These wavefunctions represent complex, high-dimensional probability distributions that scale exponentially with the number of states.

The central question of this essay is: *"How can p-bits be used to represent quantum many-body wavefunctions?"* We hypothesise that p-bits offer new and improved ways of depicting these wavefunctions, potentially outperforming classical systems in solving the quantum many-body problem. This essay will focus on two specific methods where p-bits can help to represent quantum many-body states.

The first method involves "probabilistic spin logic", a concept introduced by Camsari et al. (2017). Essentially, any logic gate function, once laid out in a truth table, can be converted into a Boltzmann-Machine style representation. Here, the probabilities of different states align with the truth table's entries. We suggest a technique to encode a known quantum state into a truth table, which is then translated into a representation using p-bits. The second method involves "neural network quantum states" (NQSs). This method uses neural networks and ML techniques to represent quantum many-body states [10]. Notably, p-bits could greatly accelerate the Monte Carlo sampling process integral to this method. This acceleration enhances both time and computational efficiency, especially when compared to standard Monte Carlo methods.

The essay is structured as follows: The first section introduces background information on p-bits and neural network quantum states. After this, we elaborate on the methods used to understand how p-bits can improve performance within the quantum many-body problem. We will then present results for both methods. Lastly, we conclude how p-bits can improve performance and discuss the results and future directions we expect for the continuation of this project. It is important to note beforehand that this research is mostly theoretical and investigative by nature since we did not have access to the needed hardware.

2 Background

2.1 Probabilistic Bits

P-bits are formally defined as binary stochastic neurons in hardware [7]. While classical bits represent either states of '0' or '1' and quantum computers have the superposition of both states, p-bits are a combination of both. A p-bit does not just represent a '0' or a '1', but it represents a probability of being in either state, as shown in Figure 1 , illustrating a clear distinction between the classical, quantum and probabilistic bits.

The coupled p-bit system is described by the following dynamical equations:

$$m_i(t) = \text{sign}\left(\tanh(\beta I_i) - r_{[-1,+1]}\right) \tag{1}$$

$$I_i(t+\delta t) = \sum_j W_{ij} m_j(t) + h_i \tag{2}$$

where m_i is the binary activation of neuron i ($m_i \in \{-1, +1\}$) [1], I_i is the effective field, r is a uniform random number drawn from the interval $[-1, 1]$, W is the symmetric coupling matrix between the p-bits, β is the inverse temperature and h is the bias vector [7].

It is important to note that equations (1) and (2) are essentially the same as the defining equations for Boltzmann machines [7]. Boltzmann machines are usually implemented in software that runs on standard CMOS hardware. However, p-bits allow them to be implemented directly in hardware, essentially creating a hardware version of a Boltzmann machine.

Serially iterating a network of p-bits described by these equations eventually generates samples approximating the Boltzmann distribution:

$$p_i = \frac{1}{Z} \exp(-\beta E_i) \tag{3}$$

where E_i is the configuration-dependent energy:

$$E(m_1, m_2, \ldots) = -\left(\sum_{i<j} W_{ij} m_i m_j + \sum_i h_i m_i\right) \tag{4}$$

which can be useful for probabilistic sampling and optimisation [11].

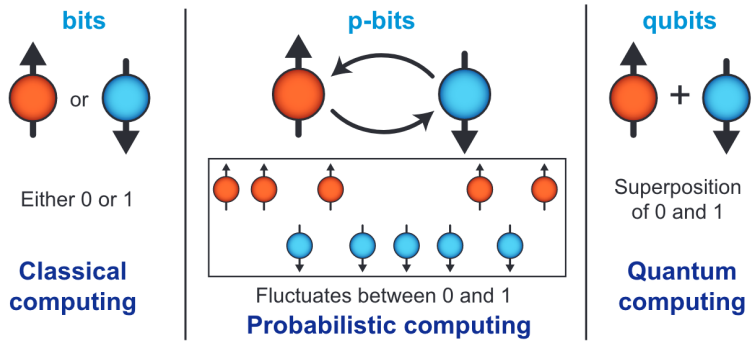

Figure 1: Figure showing the difference between classical bits, p-bits and quantum bits. Figure from [11].

[1] In physical implementations, it is often more convenient to represent p-bits as binary variables, $s_i \in \{0, 1\}$ using the transformation $m \to 2s - 1$.

2.2 Neural Network Quantum States

A primary challenge in quantum research is determining the wavefunction $\psi(s)$, encompassing all information about a quantum system. This wavefunction can be seen as a computational black box: given an input many-body spin configuration s, it outputs the amplitude and phase. However, computing the wavefunction precisely is often impractical due to the exponential growth of Hilbert space with the number of particles. Consequently, representing a generic many-body quantum state requires an exponential amount of information.

To circumvent this challenge, the wavefunction $\psi(s)$ can be approximated using a neural network, a method known as neural network quantum states, first introduced by Carleo et al. in 2017 [12]. NNQS exploit the universal approximation and dimensionality reduction capabilities of neural networks to effectively represent quantum systems, requiring significantly less information than the Hilbert space's full capacity [13]. While various neural network structures could theoretically be used for NNQS, we will focus on the Restricted Boltzmann Machine.

Restricted Boltzmann Machines

A Restricted Boltzmann Machine is an unsupervised generative model for modelling complex probability distributions, such as quantum many-body wavefunctions. Once trained, it can generate samples from these distributions, allowing it to infer unknown features of the probability distribution.

The RBM architecture comprises two layers: a visible layer with N nodes, each representing a physical spin variable ($S = \sigma_1^z, \ldots, \sigma_N^z$) in the sigma-z basis, where spins assume binary values $\sigma_i = \{-1, 1\}$, and a hidden layer of M auxiliary binary spin variables $h = (h_1, \ldots, h_M)$ with $h_i = \{-1, 1\}$. These layers are connected by symmetric weights, without any connections within a layer. It is useful to define the amount of hidden units according to a hidden unit density α: $M = \alpha N$. The structure of this network is illustrated in Figure 2.

The energy of a configuration (S, h) in the RBM is defined by:

$$E^{\mathrm{RBM}}(\mathcal{S}, h, \mathcal{W}) = \sum_j a_j \sigma_j^z + \sum_i b_i h_i + \sum_{ij} W_{ij} h_i \sigma_j^z, \tag{5}$$

with the network parameters $\mathcal{W} = \{a, b, W\}$ determining the networks response to input an input state $(\mathcal{S}, h)$ [10].

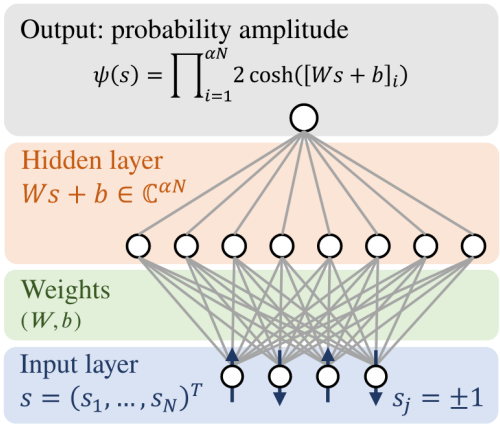

Figure 2: RBM Network Architecture with hidden-unit density $\alpha = 2$. Figure taken from [14]

RBM Training
To model our quantum system, we use the RBM to form a variational ansatz:

$$\psi_M(\mathcal{S}, \mathcal{W}) = \sum_{\{h_i\}} e^{\sum_j a_j \sigma_j^z + \sum_i b_i h_i + \sum_{ij} W_{ij} h_i \sigma_j^z}, \tag{6}$$

The primary objective of a RBM in quantum systems is to identify parameters $\mathcal{W} = \{a, b, W\}$ that minimise the system's quantum energy. The input layer of the RBM, consisting of nodes σ_i, initially represents 'quantum bits' as classical binary units. The hidden units h_i are designed to evolve and represent quantum behaviour, learning quantum correlations directly[2]. The RBM is essentially learning to reconstruct its inputs (the spin configurations) such that they follow the same statistical properties as the quantum system would.

The quantum energy of the system, given by the expectation value of the Hamiltonian, is described as:

$$E^{\mathrm{QM}}(\mathcal{W}) = \langle \hat{H} \rangle = \frac{\langle \psi_M | \hat{H} | \psi_M \rangle}{\langle \psi_M | \psi_M \rangle} \tag{7}$$

To find the optimal solution, where $\nabla E(\mathcal{W}^*) = 0$, various optimisation methods can be employed. This works uses the Stochastic Reconfiguration (SR) method, an enhanced form of gradient-descent optimisation specifically designed for quantum systems [16], which is described in Appendix A. Training the RBM using the SR method optimisation supplemented with Monte Carlo sampling is also referred to as 'Variational Monte Carlo'.

2.3 Monte Carlo Sampling

The main goal of Monte Carlo sampling for NNQSs is to effectively draw samples from the distribution $|\psi(\mathcal{S})|^2$. These samples can be used to estimate the expectation values used in the SR method, which are necessary for providing the parameter updates at each iteration.

Any general expectation value can be computed as the Monte Carlo average of the local quantity $\mathcal{O}_{\mathrm{loc}}(x)$:

$$\langle O_k \rangle = \frac{\sum_x |\psi_M(\mathcal{S})|^2 O_k^{\mathrm{loc}}(x)}{\sum_x |\psi_M(\mathcal{S})|^2} \tag{8}$$

with

$$O_k^{\mathrm{loc}}(\mathcal{S}) = \frac{\langle \mathcal{S} | \hat{O} | \Psi_M \rangle}{\langle \mathcal{S} | \Psi_M \rangle} \tag{9}$$

The Monte Carlo average can be obtained using software-based Gibbs sampling (described in Appendix B) or by using a probabilistic computer implemented with p-bits. This probabilistic computer essentially performs Gibbs sampling, but directly in hardware rather than in software. This hardware implementation can potentially offer significant increases in speed and efficiency for these kinds of calculations.

[2]Direct learning of quantum correlations requires a quantum Boltzmann machine, which is based on a different mathematical framework and utilises quantum bits for simulation. For an introduction, see Amin et al. (2018) [15]

3 Methods

Our research focused on emulating p-bits using conventional CMOS-based hardware rather then using specialised p-bit hardware. Despite the potential advantages of p-bits in terms of speed and efficiency when implemented directly in hardware, our project was limited by the lack of access to such hardware [17].

3.1 P-bits for Representations

One of our interests was how p-bits can be used to represent logic functions and quantum states. Considering that p-bits are a relatively new concept with much still to be understood, our results could help expand the realm of possibilities. We specifically investigated the representation of Boolean logic functions using p-bits, guided by the methodologies proposed by KY Camsari [7].

This work demonstrates that networks of stochastic p-bits, or probabilistic computers, can accurately implement precise Boolean functions. It describes a method for implementing any truth table directly using Boltzmann machines, without the need for learning processes. This approach of using p-bits to realise logic functions is termed "probabilistic spin logic." In the following section, we will explain how this procedure can be used to implement a XOR gate.

3.1.1 XOR Gate

Using Truth table to construct J matrix
In his paper, Camsari [7] proposes a mathematical approach to transform any truth table into a symmetric connection matrix J that will represent the given truth table in a Boltzmann machine. The basic steps for this procedure are as follows [3]:

- Write down the truth table, see table Truth table.
- Transform the binary values to bipolar values, see table Magnetised.
- Introduce new columns, consisting of four auxiliary bits and one handle bit.
- The J matrix can then be calculated with the following equation: $\sum_i u_i u_i^T$, where u_i refers to one row of the U matrix from table U matrix
- Divide the J-matrix by one common value to create an identity-like matrix with values between -1 and 1.
- Set the diagonal to zeros.

This yielded the following J matrix:

[3]Code for this procedure and sampling from the subsequent distribution can be found in our Github: `https://github.com/d-mihaila/Emulating-Small-Quantum-Systems-using-P-Bits`

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: Truth table

A	B	C
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Table 2: Magnetised

D	E	F	G	Z	A	B	C
-1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	1	-1	1	1
-1	1	-1	1	1	1	-1	1
1	-1	-1	1	1	1	1	-1

Table 3: U matrix

$$J_{XOR} = \begin{pmatrix} 0 & 0 & 0 & 0 & -1 & 1 & 1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{10}$$

Representing the XOR truth table with p-bits is equivalent to representing a two-spin triplet state, as we will explain next.

Connection to Two-Spin Triplet State
In quantum mechanics, a two-spin system can exist in either a singlet state or one of three triplet states. The triplet states are defined by having a total spin of 1, represented as either both spins up ($|\psi\rangle = |\uparrow\uparrow\rangle$), both spins down ($|\psi\rangle = |\downarrow\downarrow\rangle$), or in a mixed state ($|\psi\rangle = \frac{1}{\sqrt{2}}|\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle$)) spins. These states demonstrate a correlation between the spins without them necessarily being aligned in the same direction.

The binary values in the XOR gate can be mapped to spin states, with 0 and 1 corresponding to spin-down and spin-up, respectively. The XOR output being 1 when the inputs differ mirrors the mixed spin states of the triplet, while an output of 0 (same inputs) represents the aligned spin states.

Using 'J' matrix to construct P-bits
The J matrix can be used to initialise a system of P-bits. We received simplistic code from J. Mentik in collaboration with K. Camsari. This code contained an arbitrary so-called J matrix which we refined to represent truth tables as described in section Using Truth table to construct J matrix. The pseudocode in Algorithm 1 shows how the code used multiple combinations of 3 spin systems, two inputs and one output, to test the p-bits. For each of these systems the exact energy of the systems was computed based on a J matrix which implicitly represented the rules of the system. Furthermore the probability of each system occurring was calculated based on said energy according to the standard Equation 3, as shown in Algorithm 2. Additionally, we received code to simulate these calculations which used the same overall structure as exact Boltzmann, however, it used the equations for the representation of p-bits described in section 2.1.

$$E = -\frac{1}{2} * m^T J m + h^T m \tag{11}$$

Algorithm 1 Calculate exact Boltzmann benchmark

Require: extended truth table matrix J, number of p-bits NM
  if NM $<$ 12 **then**
    for i in range(NM**2) **do**
      binary string $\leftarrow$ generated binary string of width NM
      $m_i \leftarrow$ Magnetised version of binary string
      $E_i \leftarrow$ Computed energy of the string as a function of the magnetised binary string and J
      $Pe_i \leftarrow$ Probability as a function of energy
    end for
    return normalised Pe

Algorithm 2 Calculate probabilistic Boltzmann output

Require: number of loops NT, number of P bits in a system NM
  for i in range(NT) **do**
    for j in range(NM) **do**
      $I_i \leftarrow$ synapse equation for j-th p-bit
      $m_i \leftarrow$ neuron equation for j-th p-bit as a function of I
    end for
    k $\leftarrow$ bins ▷ given a state of the spins find out the bin number in the 2^{NM} space
    Pr $\leftarrow$ increment the corresponding bin
  end for
  return normalised Pr =0

'Brute Force' for Boltzmann Machine
In order to get the J matrix to initialise the P-bits we need to calculate the U matrix as was mentioned in section 3.1. We attempted multiple methods for constructing the U matrix as shown in table 3, the first empirical method was coined 'Brute force', this method assumed the handle bit to be 1 and attempted multiple combinations of auxiliary bits, accepting the combinations such that $U^T U$ could be inverted.

Hopfield network for Boltzmann machine
A Hopfield network was created to improve the search efficiency for the correct pattern of auxiliary bits in the U matrix for the Boltzmann machine, as proposed in 'Brute Force' for Boltzmann Machine.

A Hopfield network is a fully connected neural network with bidirectional connections. This network has an interesting characteristic, it can implement associative memory. The memory is created by encoding memory states in the weights. This ability makes it possible for the model to correct incorrect or corrupted initial data [18].

Our Hopfield network implementation uses binary and bipolar states. The bipolar states are memory states used for the weight matrix acquisition, an example of such a bipolar state is shown in Equation (12). In this manner, we represent all true state on the XOR truth table. The first bit is used as a handle bit for symmetry. The weight matrix is acquired according to Equation (13) with the constraint in Equation (14).

The computation of the model is described in Algorithm 3. This pseudocode substitutes the incorrect data in x to y and perturbs each bit towards the closest one in proximity to the memory state, in the associative memory of the weight matrix.

$$state_1 = \begin{pmatrix} 1 & -1 & 1 & 1 \end{pmatrix} \tag{12}$$

$$W = \begin{bmatrix} state_1 \\ state_2 \\ state_3 \\ state_4 \end{bmatrix} \begin{bmatrix} state_1 & state_2 & state_3 & state_4 \end{bmatrix} \tag{13}$$

$$W_{ii} = 0 \tag{14}$$

The associated memory represents the XOR gate with bipolar numbers, similarly to the U matrix. The difference lies within the transformation applied to the truth table, where Hopfield multiplies with the transposed truth table, Boltzmann adds auxiliary bits. Unfortunately, the Hopfield matrix could not substitute the 'Brute force' method inferred from Camsari [7] in the Boltzmann Machine. Our U matrix was numerically constrained, needing -1 or 1 everywhere, described above in section P-bits for Representations, while Hopfield generates a scaled binary matrix.

Algorithm 3

Require: Memory state matrix MM, binary memory state bMM matrix
  for k in range(length(MM)) **do**:
    weight $\leftarrow MM_k^T \cdot MM_k$
    y = x
    for i in range(length(x)) **do**:
      if $x_i + y@weight_{,i} > 0$ **then** :
        $y_i = 1$
      end if
      if $x_i + y@weight_{,i} < 0$ **then**:
        $y_i = 0$
      end if
      if y **then** $== bMM_k$
        break loop
      end if
    end for
  end forreturn y

Additionally, we explored the Hopfield network through its energy (Equation (15)) and probability outputs (Pe as shown in algorithm 1), to get a better understanding of how the XOR gate was represented in this network. We used the energy (equation 16) and probabilities (algorithm 1) from the 'Brute force' representation of the XOR gate, matrix U, as the memory matrix described in 13. In contrast, with the Boltzmann machine restrictions mentioned earlier in this section, this reverse implementation from Boltzmann to Hopfield was permitted by the Hopfield network.

$$E = -\frac{1}{2} y^T W y \tag{15}$$

$$E = -\frac{1}{2} * m^T J m + h^T m \tag{16}$$

To conclude this section, we were able to use brute force to find any representation of a logic gate with our implementation of p-bits, but we were not able to find a way to not brute force this and still guarantee an exact solution. Additionally, this representation does not extend beyond logic gates, so it is rather limited. In the next section we will be experimenting with a representation that does not have the limitation of logic gates.

3.1.2 MCMC-inspired

Using W matrix to construct J matrix
An alternative method that was proposed was to use the W matrix from MCMC sampling to construct a J matrix. This matrix would already be known before starting sampling, so it can be extracted and used to compute J. As can be seen in the J matrix in equation 17, it can be structured in such a way that the top left and bottom right parts of the matrix are all 0 and W and W^T are used to fill in the remaining space. This structure resembles the weights of a Restricted Boltzmann Machine. This gives us the possibility to use the weights used in MCMC sampling to construct our J matrix.

$$J = \begin{pmatrix} 0 & W^T \\ W & 0 \end{pmatrix} \tag{17}$$

Experimentation with various random W matrices shows that we can represent a wide variety of different distributions in this way. One such an example is shown in figure 6, which was

created with a W matrix as shown in equation 18.

$$W = \begin{pmatrix} -1 & 1 & -0.3 & -1 \\ -0.5 & -1 & -1 & 1 \\ -1 & -0.6 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{pmatrix} \tag{18}$$

3.2 P-bits for Sampling

In this subsection, we explore the application of probabilistic bits (p-bits) for the purpose of accelerated sampling in small quantum systems. Specifically, we describe our approach to implementing Neural Network Quantum States (NNQSs) through a Restricted Boltzmann Machine (RBM). Although our original intention was to compare the performance of p-bits with traditional Gibbs sampling methods, the lack of specific hardware restricted us to a purely software-based approach. Nevertheless, we present these software-generated results to demonstrate the workings of NNQSs and to highlight the potential areas where p-bits could significantly enhance the speed and efficiency of the process.

3.2.1 Simulating small quantum systems

We implemented a NNQSs using a RBM to determine the ground state of an antiferromagnetic 4x4 Heisenberg Model. The Hamiltonian of the system is given as:

$$\hat{H} = J_{\text{ex}} \sum_{\langle ij \rangle} \hat{S}_i \cdot \hat{S}_j \tag{19}$$

where $J_{\text{ex}} = 1$, $\hat{S}_i$ are spin variables and $\langle ij \rangle$ is used to denote summation over nearest neighbours.

Our implementation made use of the ULTRAFAST package [19] and is written in the Julia [20] programming language. The psuedocode Algorithm 4 details the broad-view implementation, omitting specific optimisation details. We compared the obtained results with exact results obtained using exact diagonalisation with the NetKet library [21].

Algorithm 4 Training RBM using Variational Monte Carlo

Require: Number of visible nodes N, number of hidden nodes M, hidden unit density α, learning rate η, Hamiltonian $\hat{H}$

```
Initialise RBM parameters W = {a, b, W}
Prepare initial spin configuration S
while not converged do
    for k = 1 to number of Monte Carlo steps do
        Perform Gibbs sampling to obtain S^(k)
        Calculate ψ_M(S^(k)) using Eq. (6)
    end for
    Compute SR method expectation values and update W
    Evaluate quantum energy E^QM(W) using Eq. (7)
    Check for convergence
end while
return Optimised parameters W
```

3.2.2 MCMC: Implementation for sampling

Experimentation with actual W matrices extracted from existing systems was not successful. An attempt was made to replace the sampling of code from the ULTRAFAST package.

However, the sampling used a lot of parameters that were in different formats and were not directly translatable to use for our P-bit implementation. Additionally our implementation was not faster than MCMC sampling. Even if we were able to implement our p-bits into ULTRAFAST, it will not improve in terms of speed. One of the causes of the slow speed is that we are working with a simulation, meaning that we still need some kind of sampling function. Having the correct hardware would solve this issue.

4 Results

4.1 P-bits for Representations: XOR Gate

4.1.1 'Brute Force' for Boltzmann Machine

One of the acceptable patterns found through 'Brute Force' for Boltzmann Machine is shown in a U matrix in table 3. This was manageable through brute force since the space of the handle bits is still manageable. The space is 4 by 4, so the worst case scenario would be to have to try 65536 combinations, as we would have $2^{4\times4}$.

When running our simulation with the J matrix described in P-bits for Representations we received probabilities for A, B, and C that represent an XOR gate (shown in figure 3). The method as described in the above code led to generalisable code for any logic gate with two inputs and one output. As part of the solution includes brute forcing, we were not able to scale this to bigger systems.

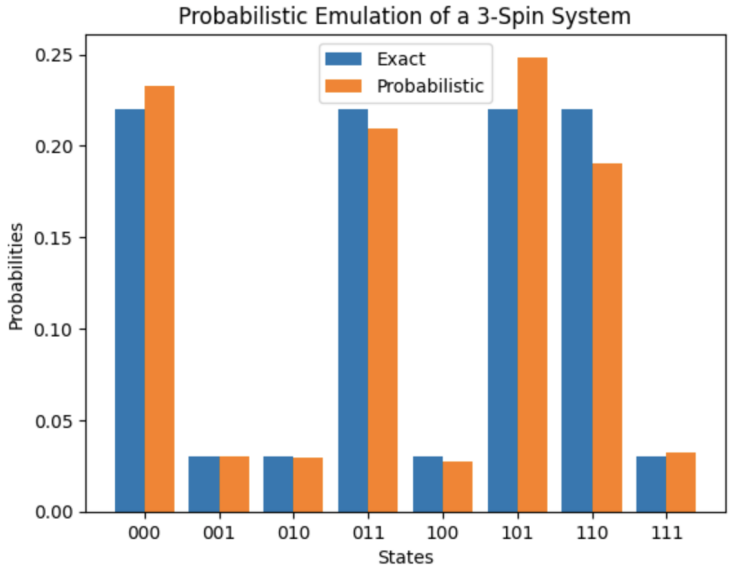

Figure 3: Probabilities of an implementation of an XOR gate

4.1.2 Hopfield network for Boltzmann machine

We found a distinct difference between the Boltzmann machines' outputs and those of the Hopfield network.

When plotting the Boltzmann machine and the Hopfield network, we found the following results, Hopfield is less stable than Boltzmann when representing the XOR gate in probabilities and Boltzmann oscillates around an energy of 0 while Hopfield operates in more negative energy. This could indicate that Hopfield, although it yields proper results less stable and

energy efficient than Boltzmann. Alternatively, the Hopfield network could output correct data but not be suited for the 'forced' matrix discussed above.

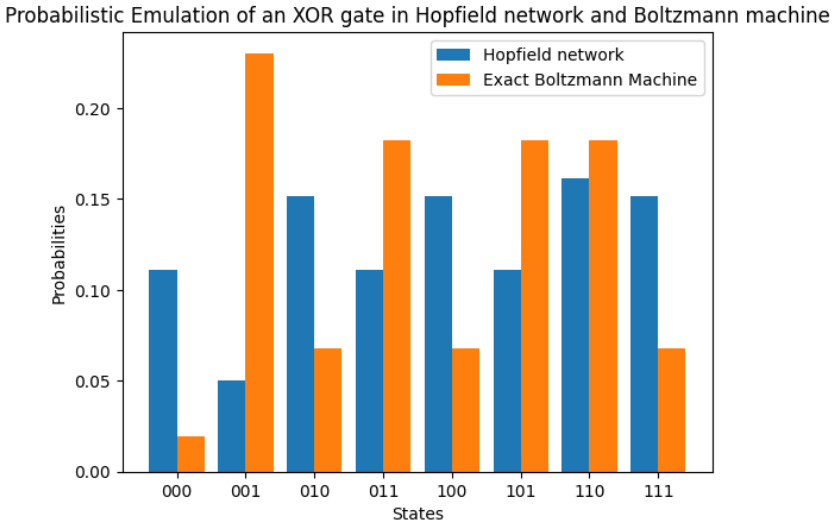

Figure 4: Probabilities of an implementation of an XOR gate in a Hopfield network and a Boltzmann machine

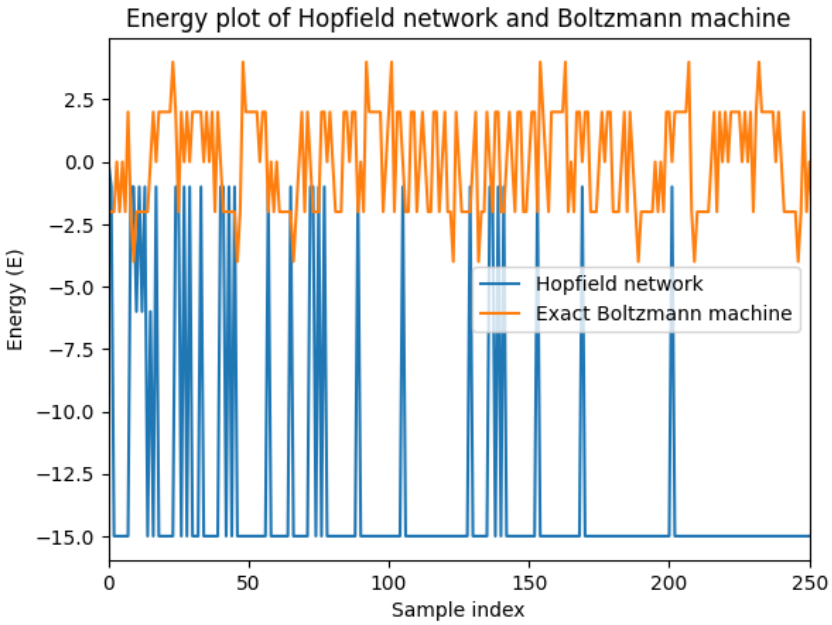

Figure 5: Energy functions of an implementation of an XOR gate in a Hopfield network (equation 15) and Boltzmann machine (equation 16)

4.1.3 MCMC: Using W matrix to construct J matrix

The MCMC method of using a W matrix structure to construct our J matrix showed some advantage over using an algorithmic approach to represent a logic gate. There was a wide variety of different possible distributions, in addition to the ability to have weights that were not strictly 1 or -1. An example of a distribution generated with this method is seen

in figure 6. This specific distribution was made with the W matrix shown in equation 18. Unfortunately we were not able to implement this representation in Julia, which would allow us to merge it into the code of the ULTRAFAST package.

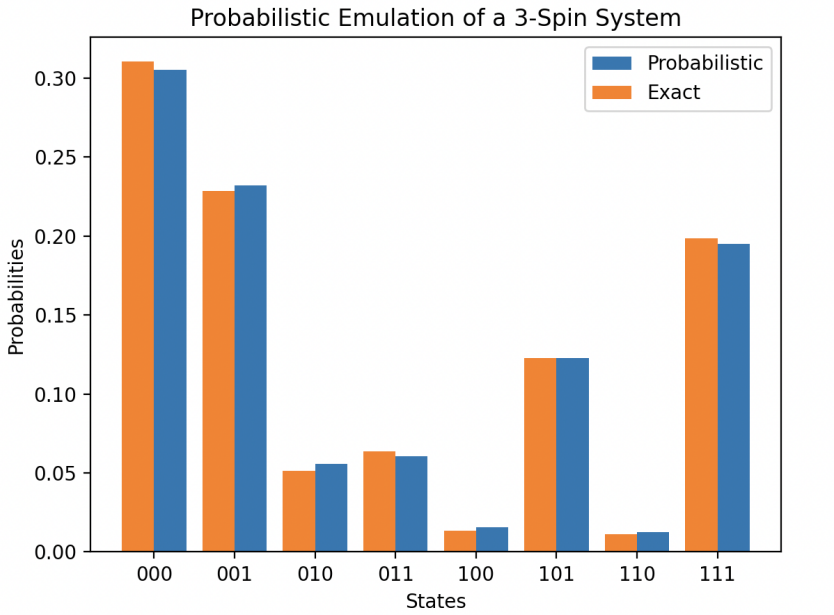

Figure 6: Probabilities of a system where a random W matrix was used to construct J

4.2 P-bits for Sampling

The RBM ansatz typically did not achieve the true ground state, except in simple cases. As a result, the estimated energy value E^0 tended to converge to a level higher than the actual ground state energy. This is shown in Figure 7. Here,the RBM was initialised with random weights and biases and hidden unit density $\alpha = 2$ and on a 4x4 lattice. 300 iterations were performed with learning rate $\eta = 0.005$, using 2000 Monte Carlo samples for each iteration.

As shown in the Figure, the energy approaches the true ground state energy E^0 in about 150 iterations. The RBM found a ground energy of $E^0_{\text{RBM}} = -44.878 \pm 1.443$. This was close to the exact results $E^0_{\text{exact}} = -44.914$. Increasing the hidden unit density improves these estimates. For example, for $\alpha = 4$, we found that $E^0_{RBM} = -44.925 \pm 0.516$.

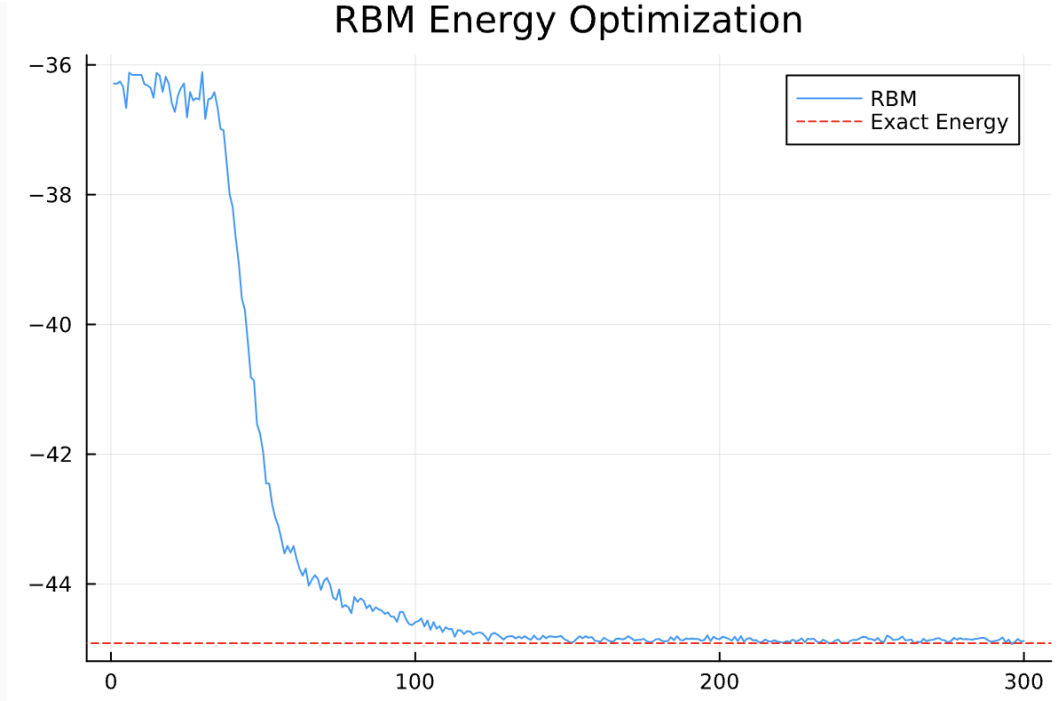

Figure 7: Ground state optimisation performed using Variational Monte Carlo on 4x4 Antiferromagnetic Heisenberg model. The RBM (blue) was initialised with random parameters, $\alpha = 2$ and $\eta = 0.005$. Exact results (red) were obtained using exact diagonalisation.

5 Conclusion

We have showcased two potential applications for probabilistic bits, to discover how p-bits can be used to represent quantum many-body wavefunctions. The first being the representation of logic gates through 'brute force', Hopfield network and Markov chain Monte Carlo inspired weight matrix methods. We found brute force to be a stable yet efficient solution compared to the other two. This method was robust at the cost of the code efficiency. Consequentially, the Hopfield network was introduced to compensate for the search inefficiency of 'brute force', however, this method came at the cost of robustness and energy stability. Lastly, we noticed a similarity between an already existing method, MCMC, and our own J matrix. Based on this we were not only able to represent bipolar logic gates, but also representations with probabilities.

We continued into a more complex application, sampling. We implemented NNQSs using a RBM that applied ULTRAFAST. This resulted in approach of the ground state around 150 iterations. This implementation was created a baseline for the same implementation with p-bits. However, when implementing MCMC-inspired W to replace the sampling in the ULTRAFAST algorithm. As mentioned in the results, this was not successful. Although we found results for the baseline implementations, these results could not lead to meaningful conclusions without results from the p-bits. So the sampling results remain inconclusive.

As we have shown in this report p-bits can be utilised to represent quantum many-body wavefunctions with respect logic gates. We managed to define numerous methods of standardising the way p-bits could represent logic gates. This clear process on how to represent logic gates with p-bits is a novelty. We attempted to push the probabilistic bits into more directions such as its role in sampling but were unable to complete this method. Future projects should delve deeper into the possibilities of p-bits for sampling.

6 Discussion

The restricted connectivity of RBMs allows for more efficient computation using CPUs due to the structure of the network, in contrast to the case of DBMs, where the hidden layers add complexity and thus are more computationally costly. Furthermore the DBM optimisation takes place over a larger set of parameters $W = (a, b, b', W, W')$ and that the expectation values required to update the parameters are harder to compute, as it is also needed to sample over two distribution instead of one each time. This increases the difficulty in implementing the DBM as opposed to RBM in classical CPU computation.

A DBM could have order of magnitudes higher precision then the results shown in Figure 7 due to its higher representational capacity. RBM with p-bits for NNQSs has already been done. Similarly, a DBM with p-bits has already been done. Our intention was to implement a DBM with p-bits for NNQSs. However, due to the complexity of this task and inaccessibility to the relevant hardware, our project is of a more exploratory and theoretical manner. We are stating the necessities and future steps needed to fulfil our intent in the future.

Representing many-body quantum states more efficiently using p-bits can have many useful applications. These include aiding condensed matter research in creating superconductors, quantum chemistry, research in Bose-Einstein condensates and Nuclear and Atomic physics. Therefore, we propose further investigation and implementation on hardware of probabilistic bits for many body quantum states.

Appendices

A Stochastic Reconfiguration

The Stochastic Reconfiguration (SR) method is an enhanced form of gradient-descent optimisation specifically designed for quantum systems. The central idea on which the SR method is build, it that a quantum system undergoing imaginary time evolution will naturally reach its lowest energy state, just like any physical system would[4]. If we can evolve the neural network representation through imaginary time, we can get a ground state representation of the wave function.

The Stochastic Reconfiguration method provides an update rule of the variational parameters such that the distance between quantum states undergoing an approximate imaginary-time evolution is minimised. The "distance" between two quantum states $(|\psi\rangle, |\phi\rangle)$ is given by the Fubini-Study metric $\mathcal{F}$ [22]:

$$F[|\psi\rangle, |\phi\rangle] := \arccos \sqrt{\frac{\langle\psi|\phi\rangle \langle\phi|\psi\rangle}{\langle\psi|\psi\rangle \langle\phi|\phi\rangle}}. \tag{20}$$

To evolve an initial quantum state $|\psi_0\rangle$ through a time step τ, we operate on the state using the matrix exponential of the Hamiltonian $\hat{H}$:

$$|\psi(\delta_\tau)\rangle = e^{-\delta_\tau \hat{H}} |\psi_0\rangle \tag{21}$$

The optimal parameter update $\mathcal{W} \to \mathcal{W} + \delta\mathcal{W}$ that minimises the distance between the state after a small imaginary-time evolution$|\psi(\delta_\tau)\rangle$ and the variational state after the parameters have been $|\psi_0 + \delta\psi_0\rangle$ is given by:

$$\delta\mathcal{W} = \arg\min_{\delta\mathcal{W}} \mathcal{F}\left[e^{-\delta\tau H} |\psi_0\rangle, |\psi_0 + \delta\psi_0\rangle\right], \tag{22}$$

Here, $\arg\min_{\delta\mathcal{W}}$ seeks the particular change in parameters that minimises $\mathcal{F}$. This expression can be simplified further analytically, but the details do not aid with understanding and are beyond the scope of this report. See [23] and the supplementary materials of [24] for a full derivation.

We obtain the following update rule for the variational parameters $\mathcal{W}$:

$$\delta\mathcal{W}_k = -\delta\tau \sum_l (S^{-1})_{kl} f_l \tag{23}$$

where $\delta\tau$ is the imaginary-time step (analogue to learning rate η in gradient descent). In this expression, we introduced the Hermitian covariance matrix S_{kl}, whose elements are given by:

$$S_{kl} = \langle O_k^\dagger O_l\rangle - \langle O_k^\dagger\rangle\langle O_l\rangle \tag{24}$$

and the components of the generalised force f_k are given as:

$$f_l = \langle O_k^\dagger H\rangle - \langle O_k^\dagger\rangle\langle H\rangle, \tag{25}$$

where † denotes the Hermitian conjugate and O_k denotes a diagonal matrix of variational derivatives, whose elements are given by:

$$O_k = \sum_{\mathcal{S}} O_k^{\text{loc}}(\mathcal{S})|\mathcal{S}\rangle\langle\mathcal{S}| \tag{26}$$

[4]In quantum mechanics, time is usually a real variable. However, some computational methods treat time as imaginary (multiplied by the imaginary unit i), transforming the quantum problem into one resembling statistical mechanics, which is often easier to solve.

in which the diagonal entries are given by the local energies:

$$O_k^{\text{loc}}(\mathcal{S}) = \frac{\delta_k \psi_M(\mathcal{S})}{\psi_M(\mathcal{S})}$$

where ψ_M is our variational ansatz for the wavefunction and $\delta_k \equiv \frac{\delta}{\delta \mathcal{W}_k}$.

The expectation values in (24) and (25) can be computed either by a full summation over the entire Hilbert space, requiring a sum over 2^N states or Monte Carlo sampling. Since full summation quickly becomes impractical, we'll concentrate on Monte Carlo sampling.

B Gibbs Sampling

Gibbs sampling [25] is designed to effectively sample from the the target distribution $P(\mathcal{S}) = |\psi_M(\mathcal{S})|^2$ using a Markov chain of samples (i.e. spin configurations) $\mathcal{S}^{(1)} \to \mathcal{S}^{(2)} \to \ldots \to \mathcal{S}^{(P)}$. The algorithm works by randomly selecting a neuron and updating it based on the conditional distribution of that neuron given the current values of all other variables, as given by equations (1) and (2).

After a certain number of steps determined by the equilibration time of the process, configuration states are sampled according to the distribution $P(\{S\})$. These samples can be used to estimate the expectation values.

It is crucial to ensure that the effective input each neuron m_i receives is computed *before* the neuron updates. This means the neurons are updated *serially* (i.e. one by one), rather then parallel (i.e. all at the same time), in order for the p-bit system to effectively converge to the Boltzmann distribution. The major drawback is that only one neuron can be updated at a time, only after the previous computation is completely done.

References

[1] C. Ostrau, C. Klarhorst, M. Thies, and U. Rückert, "Benchmarking neuromorphic hardware and its energy expenditure," *Frontiers in neuroscience*, vol. 16, p. 873 935, 2022.

[2] J. E. Niven and S. B. Laughlin, "Energy limitation as a selective pressure on the evolution of sensory systems," *Journal of Experimental Biology*, vol. 211, no. 11, pp. 1792–1804, 2008.

[3] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, 2022.

[4] T. Branco and K. Staras, "The probability of neurotransmitter release: Variability and feedback control at single synapses," *Nature Reviews Neuroscience*, vol. 10, no. 5, pp. 373–383, 2009.

[5] S. Schug, F. Benzing, and A. Steger, "Presynaptic stochasticity improves energy efficiency and helps alleviate the stability-plasticity dilemma," *Elife*, vol. 10, e69884, 2021.

[6] T. Palmer, "Human creativity and consciousness: Unintended consequences of the brain's extraordinary energy efficiency?" *Entropy*, vol. 22, no. 3, p. 281, 2020.

[7] K. Y. Camsari, R. Faria, B. M. Sutton, and S. Datta, "Stochastic p-bits for invertible logic," *Physical Review X*, vol. 7, no. 3, p. 031 014, 2017.

[8] G. Carleo, I. Cirac, K. Cranmer, *et al.*, "Machine learning and the physical sciences," *Reviews of Modern Physics*, vol. 91, no. 4, p. 045 002, 2019.

[9] G. De Chiara and A. Sanpera, "Genuine quantum correlations in quantum many-body systems: A review of recent progress," *Reports on Progress in Physics*, vol. 81, no. 7, p. 074 002, 2018.

[10] G. Carleo and M. Troyer, "Solving the quantum many-body problem with artificial neural networks," *Science*, vol. 355, no. 6325, pp. 602–606, 2017.

[11] S. Chowdhury, A. Grimaldi, N. A. Aadit, *et al.*, "A full-stack view of probabilistic computing with p-bits: Devices, architectures and algorithms," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 9, no. 1, pp. 1–11, Jun. 2023, arXiv:2302.06457 [physics], ISSN: 2329-9231. DOI: 10.1109/JXCDC.2023.3256981. [Online]. Available: http://arxiv.org/abs/2302.06457 (visited on 10/18/2023).

[12] G. Carleo and M. Troyer, "Solving the quantum many-body problem with artificial neural networks," *Science*, vol. 355, no. 6325, pp. 602–606, Feb. 2017, ISSN: 1095-9203. DOI: 10.1126/science.aag2302. [Online]. Available: http://dx.doi.org/10.1126/science.aag2302.

[13] Z.-A. Jia, B. Yi, R. Zhai, Y.-C. Wu, G.-C. Guo, and G.-P. Guo, "Quantum neural network states: A brief review of methods and applications," *Advanced Quantum Technologies*, vol. 2, no. 7-8, p. 1 800 077, 2019.

[14] D. J. Kösters, B. A. Kortman, I. Boybat, *et al.*, "Benchmarking energy consumption and latency for neuromorphic computing in condensed matter and particle physics," *APL Machine Learning*, vol. 1, no. 1, 2023.

[15] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and R. Melko, "Quantum boltzmann machine," *Physical Review X*, vol. 8, no. 2, p. 021 050, 2018.

[16] S. Sorella, M. Casula, and D. Rocca, "Weak binding between two aromatic rings: Feeling the van der waals attraction by quantum monte carlo methods," *The Journal of chemical physics*, vol. 127, no. 1, 2007.

[17] A. Z. Pervaiz, L. A. Ghantasala, K. Y. Camsari, and S. Datta, "Hardware emulation of stochastic p-bits for invertible logic," *Scientific reports*, vol. 7, no. 1, p. 10 994, 2017.

[18] H. Ramsauer, B. Schäfl, J. Lehner, *et al.*, "Hopfield networks is all you need," *arXiv preprint arXiv:2008.02217*, 2020.

[19] G. Fabiani and J. Mentink, *Ultrafast*, `https://github.com/ultrafast-code/ULTRAFAST`, 2023.
[20] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.
[21] F. Vicentini, D. Hofmann, A. Szabó, *et al.*, "Netket 3: Machine learning toolbox for many-body quantum systems," *SciPost Physics Codebases*, p. 007, 2022.
[22] S. Luo and Q. Zhang, "Informational distance on quantum-state space," *Physical Review A*, vol. 69, no. 3, p. 032 106, 2004.
[23] Y. Nomura, "Boltzmann machines and quantum many-body problems," *Journal of Physics: Condensed Matter*, vol. 36, no. 7, p. 073 001, 2023.
[24] Y. Nomura, N. Yoshioka, and F. Nori, "Purifying deep boltzmann machines for thermal quantum states," *Physical review letters*, vol. 127, no. 6, p. 060 601, 2021.
[25] A. E. Gelfand, "Gibbs sampling," *Journal of the American statistical Association*, vol. 95, no. 452, pp. 1300–1304, 2000.